イニシャル	年齢	性別	国籍
A・A	30歳	男性	日本

最寄り駅	JR線 東京駅	**最終学歴**	○○大学 情報工学部 情報システム学科
自己PR	React/Next.js + TypeScript を中心に、案件によってはバックエンドやインフラまで 担当するフルスタックエンジニアです。 得意なのは、プロダクトの初期立ち上げと、既存サービスの設計刷新の両方です。 直近では SaaS のフロントエンド刷新でテックリードを担当し、パフォーマンス改善や チームの開発フロー整備に取り組んでいます。 コードを書くだけでなく、要件の整理や仕様の言語化にも時間をかけるタイプです。 リモート前提のチームでも、文章と非同期コミュニケーションで成果を出してきました。		
自己実績	■ ポートフォリオ https://sample-test.dev ・自己紹介、職務経歴、技術スタック、執筆・登壇実績を一覧で掲載 ・Next.js 14(App Router)+ MDX で構築、月間 PV 約3,000 ■ GitHub https://github.com/sample-test ・公開リポジトリ 40件以上、年間コントリビューション 1,200+ ・OSS への PR マージ実績:TanStack Query、shadcn/ui ほか計5件 ■ 技術ブログ ・Zenn:https://zenn.dev/sample-test 累計記事数:42本、フォロワー約1,200名、累計 Like 2,500+ 主要トピック:Next.js App Router、Server Components、 パフォーマンス改善、フロントエンドアーキテクチャ ・Qiita:https://qiita.com/sample-test 累計記事数:18本、トレンド入り 3本 ■ コミュニティ運営・イベント主催実績 ・フロントエンド勉強会「Frontend Tokyo Meetup」運営メンバー(2022年〜) 月1回開催、参加者平均30名、connpass フォロワー約800名 ・社内テックトーク主催(前職) 月1回・累計18回開催、エンジニア組織の知見共有を推進 ※上記の URL・名称・数値は架空のサンプルです。 実案件への置き換え時は、NDA に抵触しない範囲での記載を推奨します。		

取得年月	資格名	取得年月	資格名
2026年01月	○○○○○○○○資格	2025年08月	○○○○○○資格
2025年07月	○○○○○○○○○○○○資格	2025年03月	○○○○○○○○○○○○○○資格

問題解決能力	戦略的な問題解決を行い、組織全体に影響を与える
コミュニケーション能力	複雑なアイデアや意見をわかりやすく伝えられる
自律性と責任感	他人の業務も考慮に入れ、チーム全体に貢献できる
学習意欲と成長	学んだ知識を応用し、チームに還元できる
チームへの貢献度	チーム全体をリードし、成功に導いている

プロジェクト詳細

No.1

プロジェクト名	スキルシート作成ツール	**規模**	PJ：10人 チーム：5人	**業種**	サービス	**要件定義**	**基本設計**	**詳細設計**	**製造構築**	**テスト**	**保守運用**
役割	プロジェクトマネージャー（PM）	**開発手法**	アジャイル開発			●	●	●	●	●	●

期間	**業務内容**	**使用技術**
	【要件定義】 • プロジェクトの背景と目的の理解 事業責任者・関係者へのヒアリングを通じて、「ITエンジニアのスキルや実績を可視化し、企業との円滑なマッチングを実現する」というプロダクトの目的を整理しました。 エンジニア側・企業側それぞれの利用シーンを言語化し、双方にとっての価値を明確化することで、開発判断の基準として関係者間で共有しています。 • 機能要件の定義 プロジェクト単位でのスキル登録、企業評価軸に沿ったスキルシート自動整形、PDF出力、案件検索・応募といった主要機能を、ユーザーストーリー形式で洗い出しました。 初期リリースに含める機能と後続フェーズで実装する機能を切り分け、機能仕様書として整理しています。 • 制約条件の定義 個人情報・経歴情報を取り扱う性質上、データの暗号化・アクセス制御・第三者への開示制限といったセキュリティ要件を明文化しました。あわせて、PC・スマートフォン・タブレットからの利用を前提とするマルチデバイス対応、主要ブラウザ(Chrome / Safari の最新版)での動作保証を制約条件として定義しています。 • データベース設計 ユーザー情報・プロジェクト経歴・スキルタグ・案件情報といったエンティティを洗い出し、ER図として整理しました。プロジェクト経歴は「期間・体制・役割・使用技術・成果」という構造で保持することで、スキルシート出力時の自動整形ロジックと整合させています。将来的な検索・マッチング機能を見据え、スキルタグはマスタテーブル化することで拡張性を確保しました。 • 受け入れ基準の設定	**■フロントエンド** CSS ・MUI ・TailWind HTML JavaScript ・React **■バックエンド** Ruby **■クラウド** Amazon Web Services (AWS) ・EC2 **■Server OS** Linux ・bash Windows Server ・Power Shell **■クライアントOS** Windows

各機能について「どの状態になれば完成と言えるか」をシナリオベースで定義しました。たとえばスキルシート作成機能であれば「途中保存ができる」「必須項目のバリデーションが動作する」「PDF出力時にレイアウト崩れが発生しない」など、具体的な動作条件として記述。テスト工程・リリース判定の共通基準として運用しています。

- 変更管理プロセスの確立
  要件の追加・変更が発生した際の判断フローを整備しました。変更要望に対しては、影響範囲(工数・他機能への影響・スケジュールへの影響)をテンプレート化したフォーマットで整理し、優先度を再評価したうえで意思決定する仕組みを構築しています。これにより、スコープの肥大化による納期遅延を防止しました。
- 要件定義書の作成とレビュー
  機能仕様・画面仕様・データ仕様を一元管理する要件定義書を作成しました。エンジニア・デザイナー・ビジネス側それぞれの視点でレビュー会を定期的に実施し、認識齟齬を早期に解消。レビューでの指摘事項はチケットとして追跡し、要件定義書側にも反映するクローズドループを構築しています
- コミュニケーションと報告
  定例でのスプリントレビューに加え、関係者向けに進捗・リスク・意思決定事項をまとめた報告を定期実施しました。チーム内コミュニケーションは非同期前提で設計し、議事録・意思決定ログを集約することで、リモートメンバーも含めて全員が同じ情報にアクセスできる状態を維持しています。

【基本設計】

- システムアーキテクチャの設計
  プロジェクトの目的・想定トラフィック・開発体制を踏まえ、Next.js(App Router)を中核としたフルスタック構成を採用しました。Server Components を活用してサーバ側でのデータ取得を基本とすることで、クライアント側の JavaScript 量を抑え、初期表示パフォーマンスを優先する設計としています。技術選定の判断軸として「①保守性②採用市場での技術人材確保のしやすさ③将来的な機能拡張への耐性」を明文化し、関係者と合意した上で確定しました。
- セキュリティ設計
  個人情報・経歴情報を扱うプロダクトであるため、認証・認可・データ保護を多層的に設計しました。認証は OAuth ベースでパスワードレス対応を視野に入れた構成とし、認可についてはユーザーが自身のスキルシート以外を閲覧できないよう、リソース単位でのアクセス制御を設計。通信は全面 HTTPS、保存データの暗号化、機密情報の環境変数管理など、OWASP Top 10 を参照しながら基本対策を網羅しました。
- エラーハンドリング設計
  ユーザー操作時のエラー(入力不備・通信エラー・タイムアウト)とシステム内部のエラー(DB接続障害・外部API障害)を切り分け、それぞれに対する挙動を設計しました。ユーザー向けには分かりやすい文言とリカバリ手段(再試行・下書き保存からの復元)を提供し、内部エラーは Sentry に集約して即時検知できる体制を構築。「ユーザーに何が起きたか分からない画面」を出さないことを設計原則として徹底しました。
- ユーザビリティ設計
  スキルシート作成という作業負担の高いタスクを完遂してもらうために、入力フォームをステップ式に分割し、途中保存・下書き機能を標準で組み込みました。「いつでも中断できる・いつでも戻ってこられる」を設計の前提とし、入力途中の離脱率を下げる導線を意識。PC・スマートフォン・タブレットでの利用を想定したレスポンシブ設計を行い、移動中の隙間時間にも更新できる体験を実現しています。
- プロトタイプの作成
  設計判断に迷う箇所(複雑な入力フォーム、PDF出力レイアウト、認証フロー)については、本実装に入る前に動作するプロトタイプを作成して検証しました。特に PDF 出力部分は、レイアウト崩れ・改ページ制御・フォント埋め込みなど、実装してみないと判断できない要素が多かったため、早い段階で技術的な実現可能性を確認する工程を組み込んでいます。

【詳細設計】

- モジュール設計
  責務ごとにディレクトリ・パッケージを分割し、依存関係が一方向に流れるよう設計しました。UI層・ドメイン層・データアクセス層を明確に分離し、ビジネスロジックを UI フレームワークから独立させることで、将来的なリプレイスや機能拡張に耐えられる構造としています。共通的に使われる処理はパッケージとして切り出し、再利用性と保守性を両立させました。命名規則・ディレクトリ構成・依存方向のルールはドキュメント化し、チームメンバー全員が同じ判断基準でコードを配置できる状態を整えています。
- 処理フロー設計
  ユーザー操作起点の処理(スキルシート保存、PDF出力、案件応募など)について、正常系・異常系・並行処理を含めたシーケンス図を作成しました。特にデータ更新を伴う処理では、トランザクション境界を明示し、途中で失敗した場合のロールバックやリトライ方針を設計。非同期処理が必要な箇所(PDF生成、メール送信など)はジョブキューに切り出し、ユーザー操作のレスポンスをブロックしない構成としました。
- パフォーマンス設計
  初期表示の速度を優先する観点から、Server Components によるサーバ側レンダリング・Edge ネットワークでのキャッシュ配信・画像の最適化(WebP変換、遅延読み込み)を組み合わせた構成を設計しました。データベース側では、検索・一覧表示で多用されるクエリにインデックスを設計し、N+1 問題の発生しやすい箇所には事前にバッチ取得方針を定義。継続的にパフォーマンスを監視できるよう、Lighthouse スコアと Core Web Vitals を CI に組み込み、デプロイごとに自動計測する仕組みを構築しました。
- テスト設計
  テストピラミッドの考え方を基本とし、「ユニットテストで広く・結合テストで肝心な箇所・E2Eテストで主要ユーザー導線」をカバーする戦略を設計しました。ビジネスロジックを含む関数・ドメイン層には高いカバレッジを設定し、UIコンポーネントは振る舞いテスト中心、PDF出力や認証フローなど不具合があった場合の影響が大きい箇所には E2E テストを用意。テストの目的・粒度・記述方針をドキュメント化し、チーム全体で同じ基準でテストを書ける状態を整えました。
- 依存関係の管理
  利用するライブラリの選定にあたっては、メンテナンス状況・コミュニティの活発度・ライセンス・採用実績を確認し、長期的に保守できるものを選定する方針としました。バージョン管理はロックファイルを正として運用し、定期的なアップデートチェックを CI に組み込んでセキュリティ脆弱性を早期に検知できる体制を構築。メジャーバージョンアップ時の影響範囲調査・移行計画の策定も含めて、依存ライブラリの管理を継続的な運用業務として位置付けています。

2025年01月
〜
現在

1年6ヶ月

【製造構築】

- 環境準備
  ローカル開発環境を Docker ベースで構築し、メンバー全員がコマンド一つで同じ状態を再現できるよう整備しました。データベース・モックサーバ・依存サービスをコンテナ化し、OS差異による「自分の環境では動く」問題を発生させない構成としています。初期セットアップ手順・トラブルシューティング・主要コマンド一覧をドキュメント化し、新規参画メンバーが半日以内に開発を始められる状態を整えました。環境変数の管理方針も明文化し、秘匿情報の取り扱いルールを徹底しています。
- コードレビュー
  レビュー観点を「設計の妥当性・可読性・テスト網羅性・パフォーマンス・セキュリティ」の5軸に整理し、レビューガイドラインとして明文化しました。Pull Request テンプレートに変更理由・影響範囲・テスト方針を記載する欄を設け、レビュアーが短時間で文脈を把握できる仕組みを構築。指摘の粒度を揃えるため、「Must / Should / Nits」のラベル運用を導入し、優先度のある指摘とそうでない好みの違いを区別できるようにしました。これにより、レビューが感情的な議論ではなく建設的な改善の場として機能するチーム文化を醸成しています。
- 継続的インテグレーション
  GitHub Actions を中心に CI パイプラインを構築し、Pull Request 作成時に「静的解析(Lint・型チェック)・ユニットテスト・E2Eテスト・パフォーマンス計測」を自動実行する仕組みを整備しました。マージ前に品質基準を満たしていない変更がメインブランチに入ることを防ぎ、開発スピードと品質を両立しています。ビルド時間の最適化(キャッシュ活用・並列実行)にも取り組み、フィードバックループを短く保つことでレビュー・修正のサイクルが滞らないよう配慮しました。
- デバッグと修正

macOS

■DB
PostgreSQL

■デザイン
Canva
Figma
Notion

■Tool
Docker
GitHub
Google Meet
Slack
Visual Studio

	不具合対応では、再現手順の確立 → 原因の特定 → 修正 → 再発防止策の実装、という流れを徹底しました。フロントエンド側では Sentry でエラーを集約し、発生頻度・影響範囲を定量的に把握した上で優先度を決定。バックエンド側では構造化ログを出力し、リクエスト単位でトレースできる体制を構築しています。再発防止については、修正だけでなく「同じ種類の不具合が起きない設計上の改善」までスコープに含め、根本的に解決するアプローチを取りました。 ● パフォーマンスのチューニング 「計測 → ボトルネック特定 → 改善 → 効果検証」のサイクルを徹底しました。フロントエンドでは Lighthouse・Web Vitals を継続的に計測し、LCP・CLS・INP の悪化要因を分析。バンドルサイズの削減・画像の最適化・不要な再レンダリングの抑制を段階的に進めました。バックエンドでは APM ツールでスロークエリ・遅延処理を特定し、インデックス追加・クエリ最適化・キャッシュ導入で対応。改善後は必ず数値で効果を検証し、推測ではなくデータに基づいた意思決定を行うアプローチを徹底しています。 【テスト】 ● テスト計画の策定 プロダクトの特性とリリース時期から逆算し、「どの工程で・何をどこまでテストするか」のテスト戦略を策定しました。テストピラミッドの考え方を基本とし、変更頻度の高い箇所・不具合の影響範囲が大きい箇所には手厚くテストを配置し、それ以外は軽量に保つ方針です。リリース判定の基準(必須項目のテスト通過率・既知の重大不具合ゼロなど)を明文化し、関係者間で合意。テスト工数の見積もりも実装工数とセットで計画に組み込み、テストが後回しになって品質を犠牲にする状況を防ぐ運用としました。 ● テストケースの作成 同値分割・境界値分析・状態遷移などの観点を組み合わせ、最小限のケース数で広い範囲をカバーできるよう設計しました。スキルシート作成のような入力系機能では、正常入力・異常入力(空・最大長超過・特殊文字・想定外フォーマット)・境界値を網羅したケースを用意。状態が変化する機能(下書き保存・公開・非公開など)では状態遷移図をベースにテストケースを起こし、抜け漏れがない形で整理しました。テスト観点をドキュメント化することで、メンバー間でテストケースの粒度を揃えられる状態を構築しています。 ● ユニットテスト ビジネスロジックを含む関数・ドメイン層を対象に、想定外の入力や境界値も含めた挙動を検証しました。テスト対象の責務を明確にし、外部依存(データベース・外部API)はモックに置き換えることで、純粋にロジックの正しさだけを検証できる構成としています。Arrange-Act-Assert パターンに沿った記述ルールを定めることで、テストコード自体が仕様書として読める状態を維持。カバレッジは数値だけを追わず、「重要な分岐が網羅されているか」を意識した運用としました。 ● 結合テスト API とデータベース、フロントエンドとバックエンドの連携部分を対象に、モックではなく実際の依存関係を含めた状態で動作を確認しました。データの流れが複数のモジュールをまたぐ箇所(認証・権限制御・トランザクションを含む処理)を重点的にテスト対象とし、「単体では通るが結合すると壊れる」を防ぐ設計です。テスト用データベースは本番に近いスキーマで構築し、テスト実行ごとにクリーンな状態へ初期化する仕組みを整備。並行実行しても結果がブレないよう、データの独立性を担保しました。 ● 受け入れテスト 要件定義時に設定した受け入れ基準をベースに、ユーザーシナリオに沿った E2E テストを実施しました。「スキルシートを作成して PDF を出力する」「下書きを保存して翌日に再開する」「案件を検索して応募する」など、実際のユーザー行動に近い一連の流れを自動化し、機能単体では検出できない導線上の不具合を発見できる体制を構築。手動テストでは、自動化が困難な視覚的な確認(レイアウト崩れ・印刷時の表示)を中心に実施し、自動と手動の役割を明確に分けた運用としています。 ● バックアップとリカバリテスト 不具合を発見した際は、GitHub Issues に再現手順・期待挙動・実際の挙動・影響範囲を記録し、修正後は再テストで完全な解消を確認しました。重要なのは個別修正だけでなく、不具合の発生傾向を定期的に分析することです。「どの種類の不具合が・どのモジュールで・どの工程で混入したか」を集計し、テスト計画の見直しやレビュー観点の追加にフィードバック。同じ種類の不具合を繰り返さない仕組みを継続的に改善しています。 【保守運用】 ● システム監視 プロダクトの正常性を継続的に把握するため、監視対象を「アプリケーション稼働状況・パフォーマンス・エラー発生状況・インフラリソース・ビジネスKPI」の5層に分けて設計しました。各層で見るべきメトリクス(レスポンスタイム・エラー率・CPU/メモリ使用率・主要操作の成功率など)を明文化し、ダッシュボードに集約。閾値を超えた場合は自動でアラート通知される仕組みを構築し、問題の早期発見と対応を可能にしています。「監視しているつもり」にならないよう、月次でダッシュボードの内容を見直し、不要な指標の削除・新たに必要な指標の追加を継続的に行っています。 ● インシデント管理 障害発生時の対応フローを「検知 → 初動対応 → 復旧 → 原因分析 → 再発防止」の5ステップで標準化しました。検知時には影響範囲(影響ユーザー数・機能・継続時間)を最初に確認し、復旧を最優先で進めるルールを徹底。復旧後にはポストモーテム(振り返り会)を実施し、技術的原因だけでなくプロセス・体制面の課題まで掘り下げる文化を醸成しています。ポストモーテムの記録はナレッジとして蓄積し、新規メンバーのオンボーディング時にも共有することで、組織として同じ失敗を繰り返さない仕組みを構築しました。 ● リリース管理 リリースに伴うリスクを最小化するため、デプロイの仕組みを段階的にすることを設計方針としました。新機能は最初に内部限定で公開し、問題がなければ一部ユーザーへ段階的に展開、最後に全ユーザーへ公開する流れを整備。フィーチャーフラグを用いて「機能の有効化・無効化をデプロイなしで切り替えられる」状態を維持することで、不具合発生時の影響を最小化しています。リリース手順・ロールバック手順をドキュメント化し、誰が対応しても同じ品質で実施できる体制を構築しました。 ● セキュリティ管理 セキュリティ対応を一過性のものではなく継続的な運用として位置付けました。依存ライブラリの脆弱性チェックを CI に組み込み、新たな脆弱性が公表された際に即座に検知できる体制を構築。定期的なアクセスログのレビュー・権限設定の棚卸し・認証フローの動作確認を運用ルーチンに組み込み、設定漏れや不正アクセスの兆候を早期に発見できる状態を維持しています。個人情報を扱うプロダクトとして、利用者の信頼を継続的に維持することを最優先の方針としました。 ● ユーザーフィードバックの収集と対応 利用者からの問い合わせ・要望を、サポート窓口・サービス内フォームなど複数の経路で収集し、内容を分類・集計する仕組みを構築しました。フィードバックは「不具合報告・機能要望・UI/UX 改善提案・運用に関する質問」に分類し、それぞれ異なる対応フローで処理。要望の傾向は月次で集計し、PdM と共有することでプロダクトロードマップへ反映する流れを整備しました。「対応した・しない」だけで終わらせず、なぜその判断をしたかを記録に残すことで、判断基準を組織として蓄積しています。	
成果 アピールポイント	ITエンジニア向けスキルシート作成・案件マッチングサービス「スキルログ」の 新規開発に、テックリード兼フルスタックエンジニアとして参画しました。 要件定義から実装・運用まで一気通貫で担当し、MVPを5ヶ月でリリースしています。 特に注力したのは、エンジニアの入力負担を下げるスキルシート自動整形ロジックの 設計と、企業提出に耐えるPDF出力機能の実装です。これにより、スキルシート作成 時間を3〜4時間から平均23分に短縮し、PDF出力のエラー率も12%から0.4%まで 改善しました。 パフォーマンスとSEOの両立にも取り組み、LCP 1.2秒・Lighthouse 96 を維持 しながら、リリース6ヶ月でオーガニック流入を約3.8倍に伸ばしました。あわせて エージェント向け管理画面とマッチングロジックも開発し、紹介リードタイムを 5営業日から1.5営業日に短縮しています。	

エンジニア歴：1年6ヶ月

S: テックリードレベル / アーキテクトレベル
A: 独自遂行できレビューもできる / 高度な構築や運用が可能でレビューもできる
B: 一部調べながらでも独自遂行できる / 一部調べながらでも構築・運用を独自遂行できる
C: 有識者のサポートがあればある程度遂行できる / 有識者のサポートがあれば構築・運用が可能
D: 基本的な知識はあるがOJTが必要

■ フロントエンド								
A	CSS	1年6ヶ月	B	・React	1年6ヶ月			
A	・MUI	1年6ヶ月						
A	・TailWind	1年6ヶ月						
A	HTML	1年6ヶ月						
B	JavaScript	1年6ヶ月						

■ バックエンド								
B	Ruby	1年6ヶ月						

■ クラウド								
C	Amazon Web Services (AWS)	1年6ヶ月						
C	・EC2	1年6ヶ月						

■ Server OS								
A	Windows Server	1年6ヶ月						
A	・Power Shell	1年6ヶ月						
A	Linux	1年6ヶ月						
A	・bash	1年6ヶ月						

■ クライアント OS								
S	Windows	1年6ヶ月						
S	macOS	1年6ヶ月						

■ DB								
B	PostgreSQL	1年6ヶ月						

■ デザイン								
A	Canva	1年6ヶ月						
A	Figma	1年6ヶ月						
A	Notion	1年6ヶ月						

■ ツール								
-	Docker	1年6ヶ月						
-	GitHub	1年6ヶ月						
-	Slack	1年6ヶ月						
-	Visual Studio	1年6ヶ月						
-	Google Meet	1年6ヶ月						